Distributed Learning in Realistic Virtual Environment

•••

VDL Group
Weichao Qiu, Yuan Jing Vincent Yan, Kaiyue Wu

Outline

- Background
  - Reinforcement learning
  - Realistic virtual environment
- The two-stage distributed learning architecture
  - Distribute actors to multiple machines
  - Distribute learners to multiple machines
  - The complete system
- The virtual-real arm challenge
- Conclusion

Background

The AI

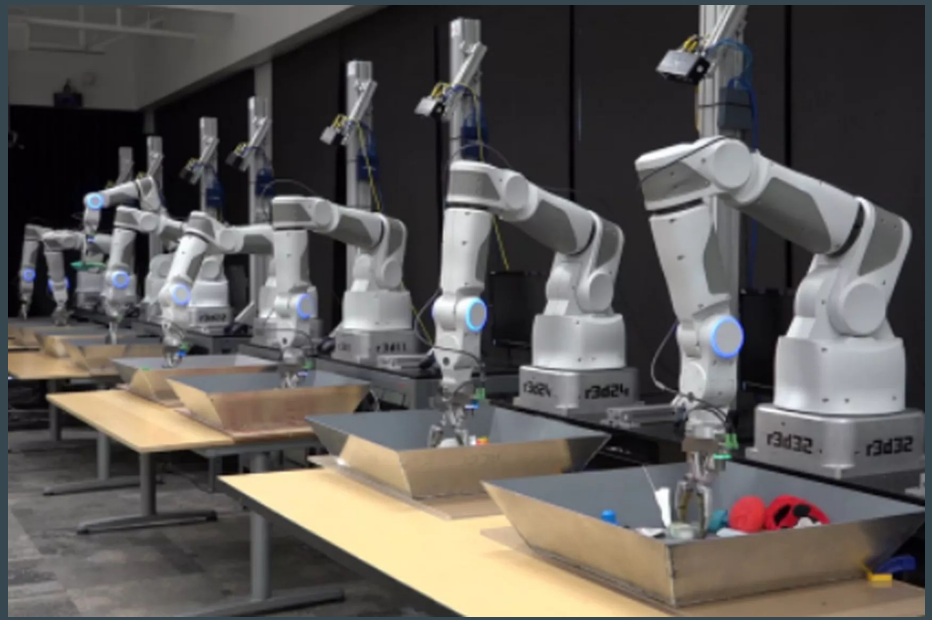
Powerful **machine learning** algorithms make it possible to teach robots to achieve complex tasks, such as flying quadcopter, walking with two legs.

The training of robots require a lot of time and efforts. The training is usually done by trial and error, which is called **reinforcement learning**.

Training with real robots

Train robot arm to grasp with reinforcement learning [Levine et al. 2016]

http://www.theverge.com/2016/3/9/11186940/google-robotic-arms-neural-network-hand-eye-coordination (from google)

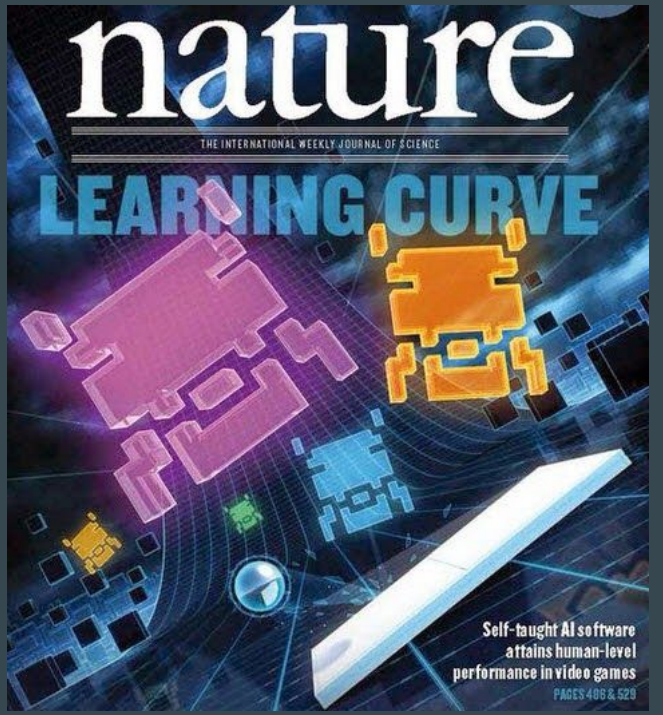


Training with real robots

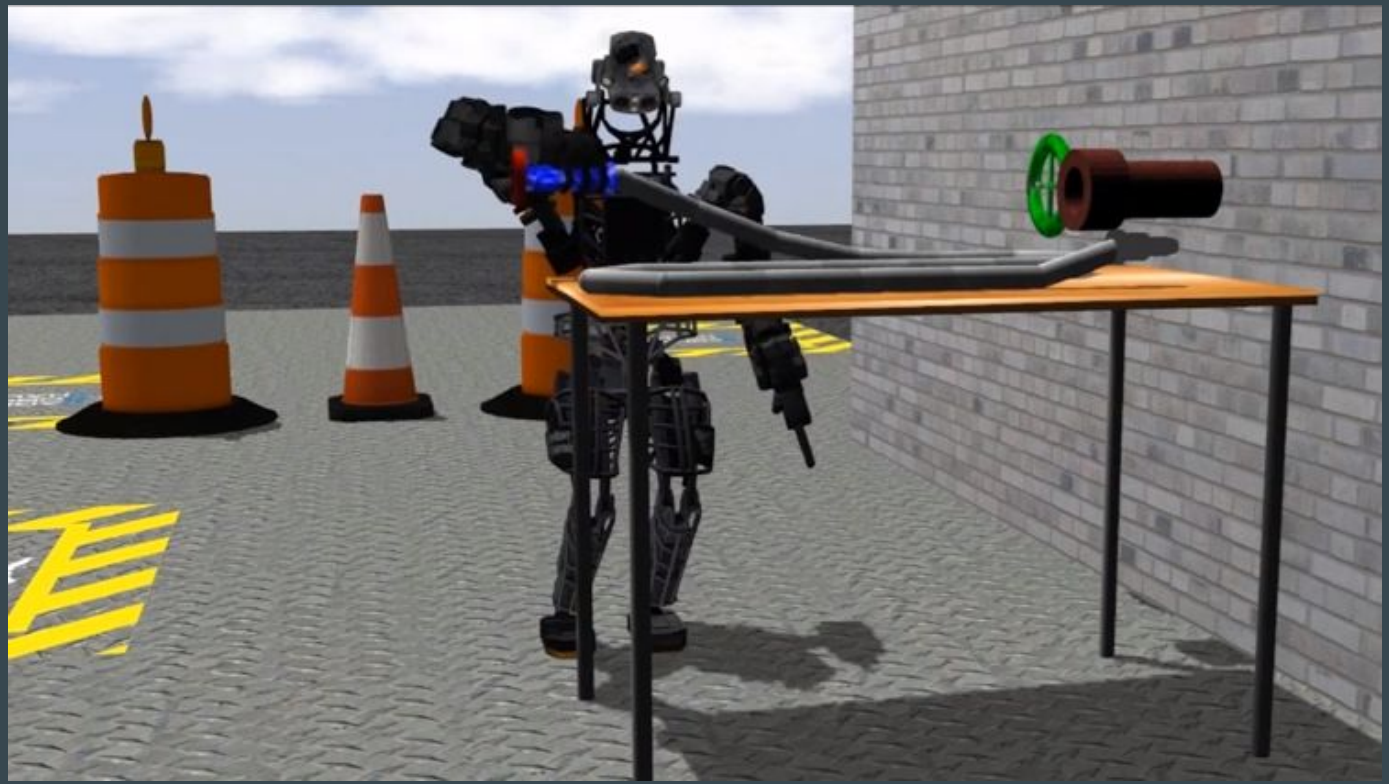
Slow and very expensive !

Background

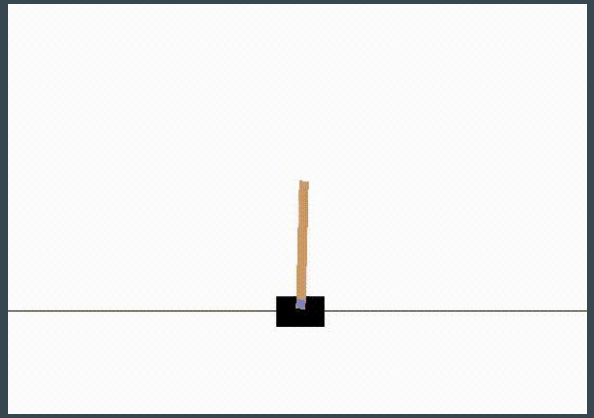
Instead of training with real robots, it is popular to do training using video game (in a virtual environment). Nature paper published by Deepmind [Mnih et al., 2015].



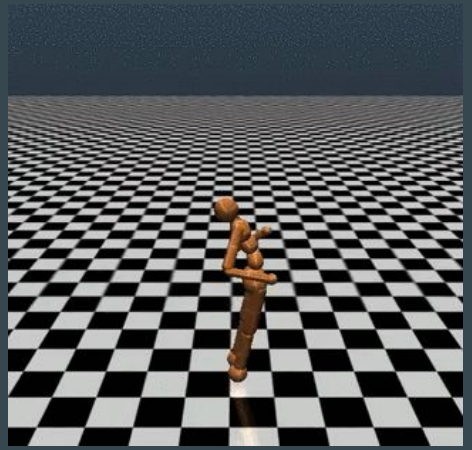
But virtual environments have very different quality



The challenges from using realistic virtual environments



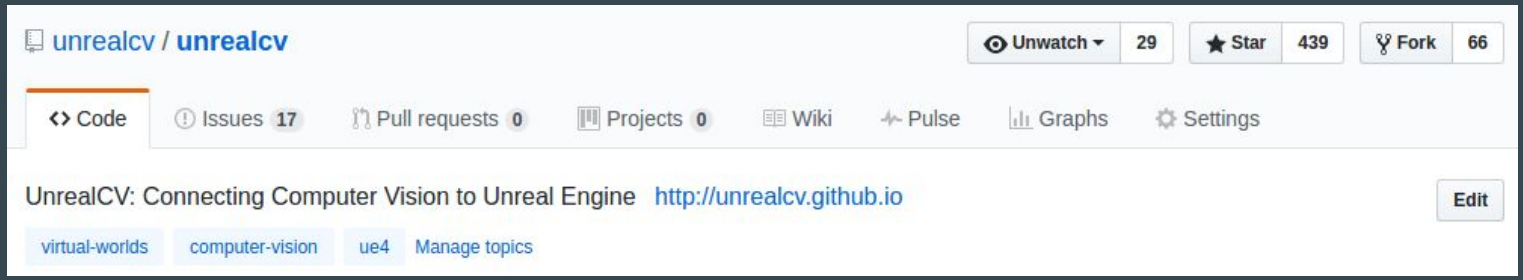
CartPole:
Memory: 560KB
FPS: 170563



Humanoid-v1:
Memory: 81280KB
FPS: 1206

RealisticRendering:
Memory: 2152132K (~2GB)
FPS: 200 (without physics simulation)

Motivation from UnrealCV



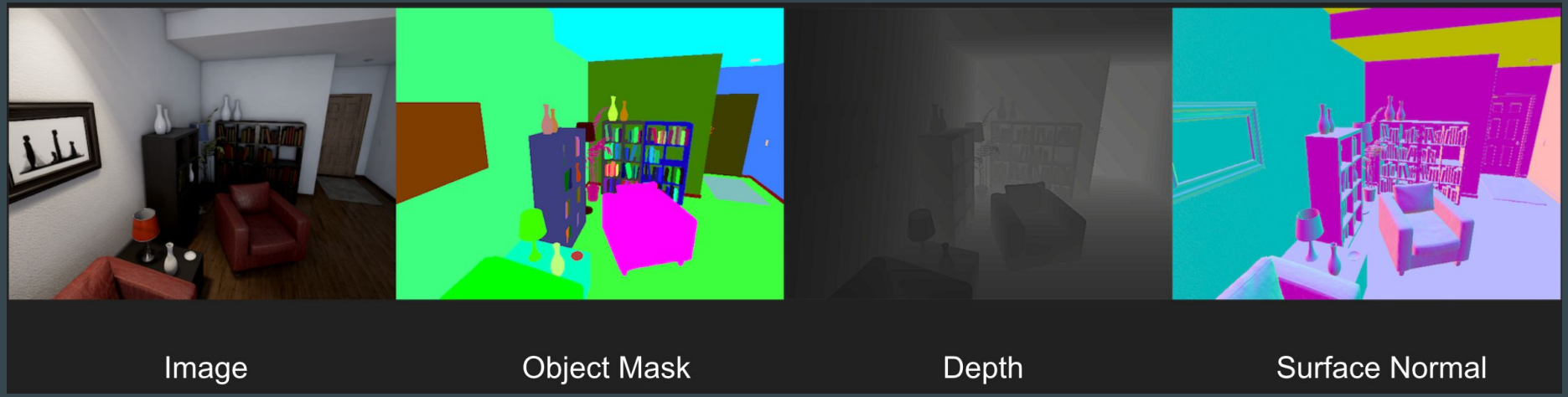


Image | Object Mask | Depth | Surface Normal

Tools we used

- **Tensorflow**
  Machine learning library in Python
- **OpenAI gym + universe**
  Virtual environments for reinforcement learning
- **Unreal Engine + UnrealCV**
  Realistic virtual environments

The NeonRace virtual environment

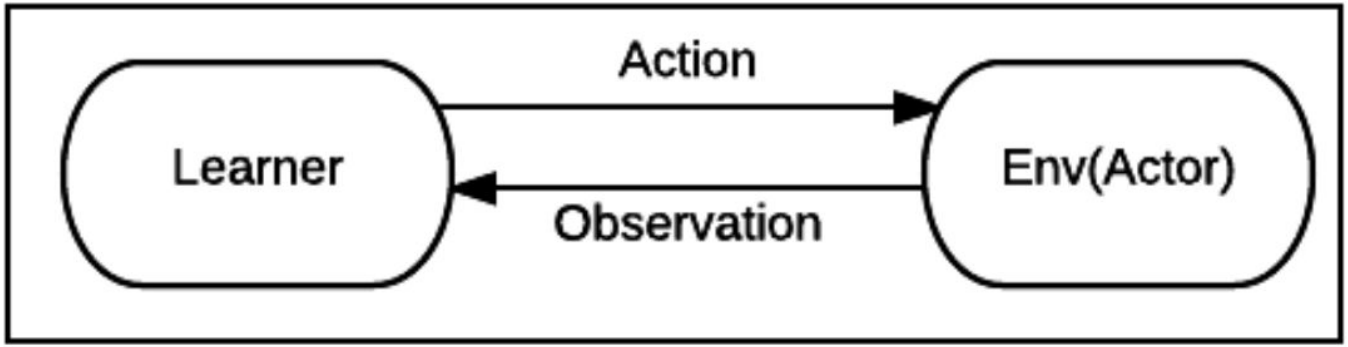
5 Frames per second
Actions:
[Stop, Left, Right, Up, Top Left, Top Right, Back]
Make decision based on the image

Two-stage distributed learning architecture

Background of Technical Details

- Learner: The program which is running the learning algorithm. It processes the data generated after applying actions to the environment.
- Actor: The program which is running the interactions with environment. It generates the data for the learner program to process.

Original Architecture



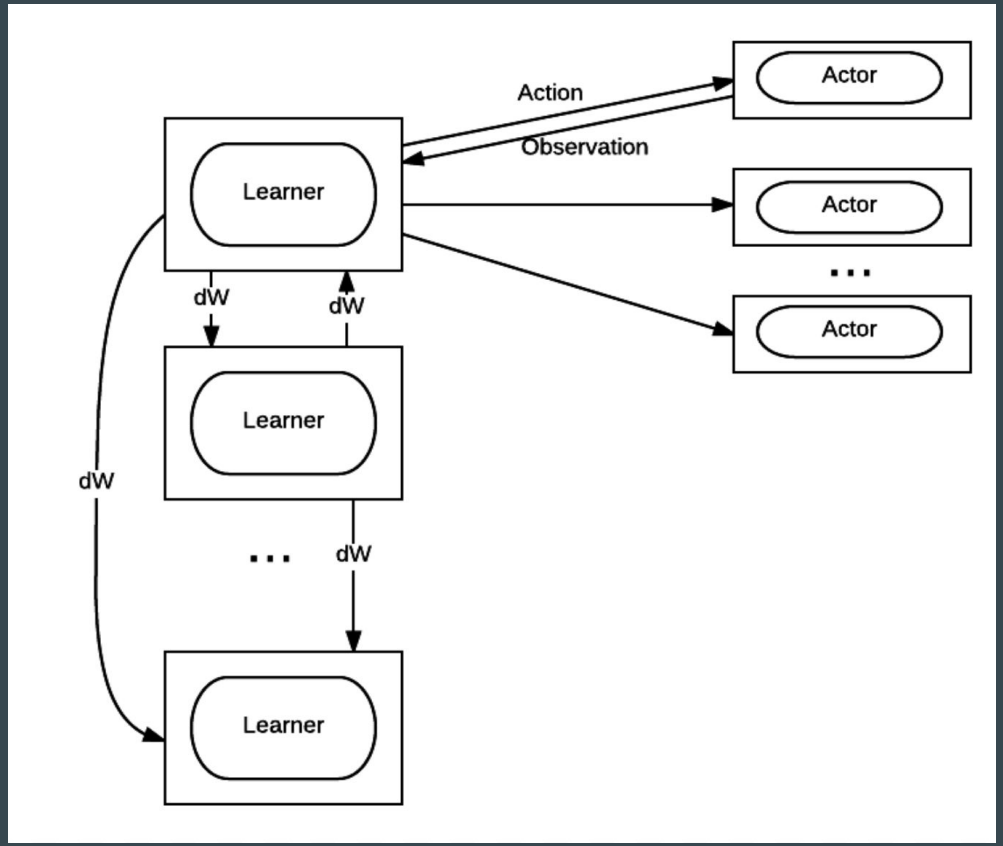
Problem

- One-learner and one-actor system is limited by the resource of a single machine.
- Learner can be parallelized to multiple learners working at the same time
- Actor can be separated out of the original learner-actor program and be parallelized to multiple actors working at the same time for each one of the learners

Problem

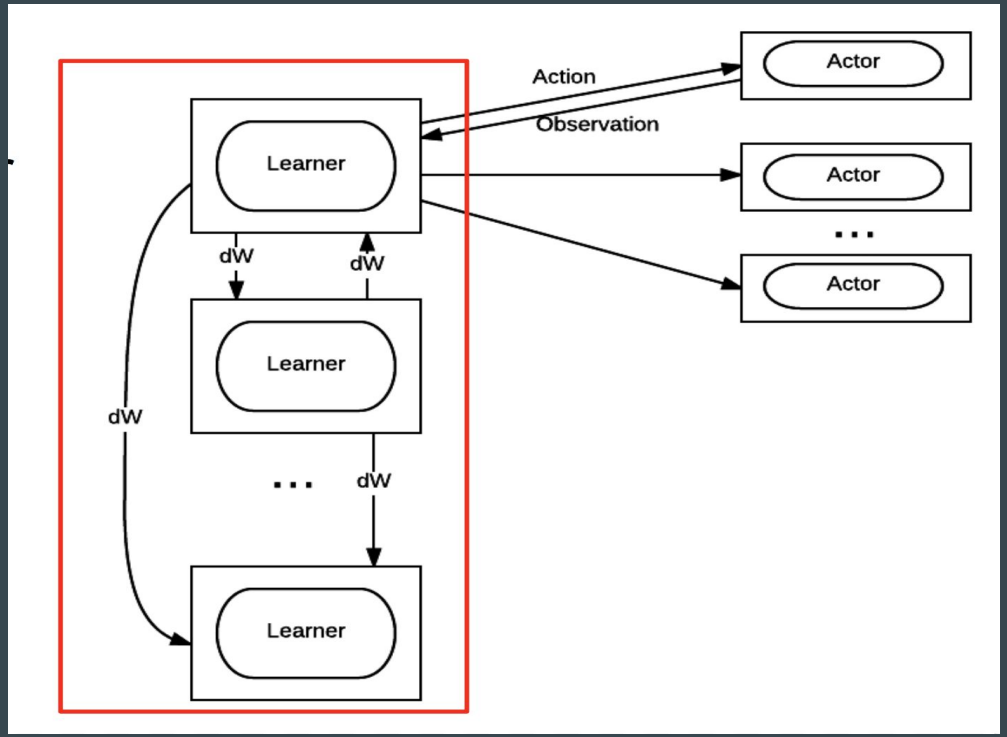
- Motivation for separation: Learners usually work a lot faster than the actors, especially in the virtual realistic environment that we are targeting, which means a learner actually has the capability to work with multiple actors at the same time.
- The original system is not making full use of the capability of the learners.

Our Architecture



Our Architecture

- Learner-learner communication

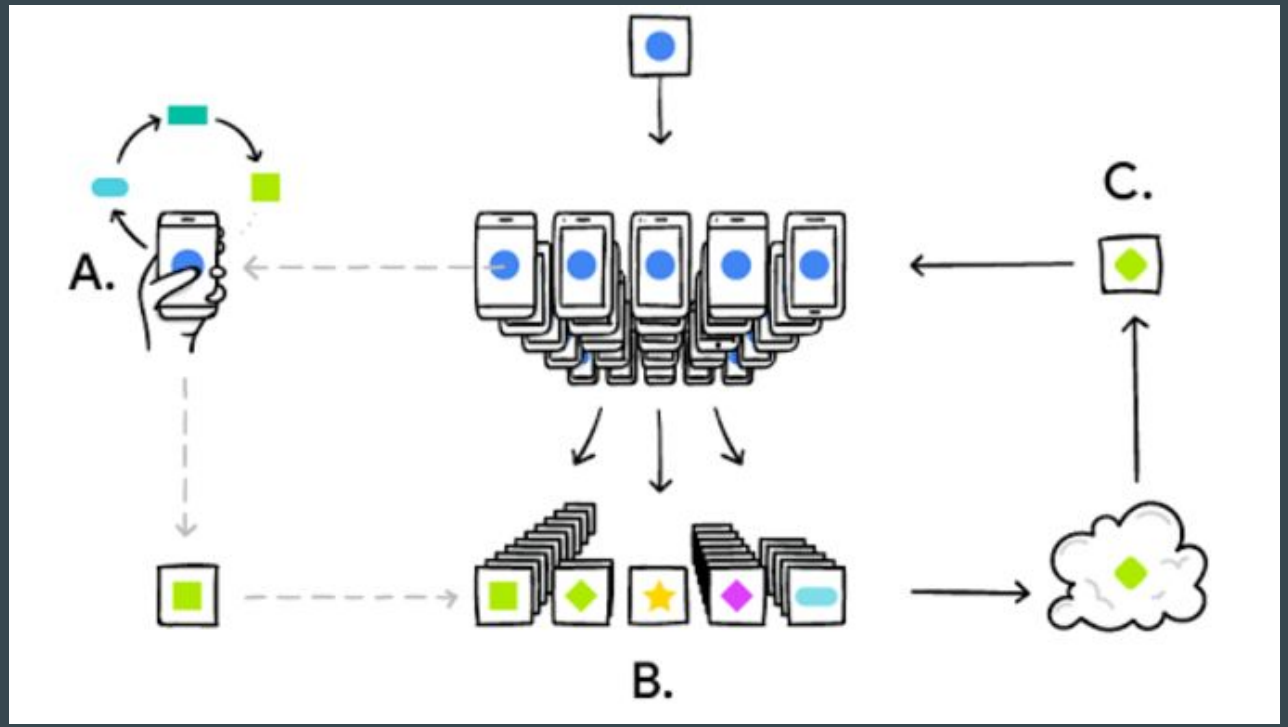


P2P Advantages

- Decentralized
- Fault tolerant (progress when partitions and crashes)
- Intrusion Tolerant (BFT protocols, low latency requirements)
- Self-scalability
- Cost effective

P2P Use Case 1

P2P Use Case 2

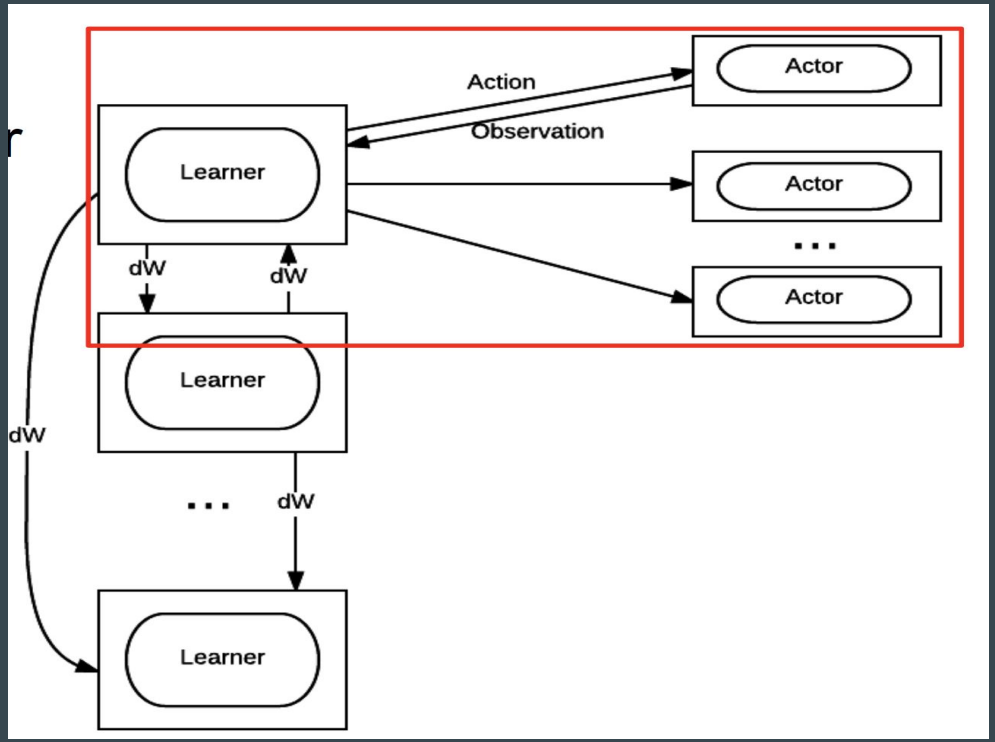


“Federated Learning”

Multi-Actor Training

Our Architecture

- Learner-actor communication



Question:

- Can any task be sped up by the separation?

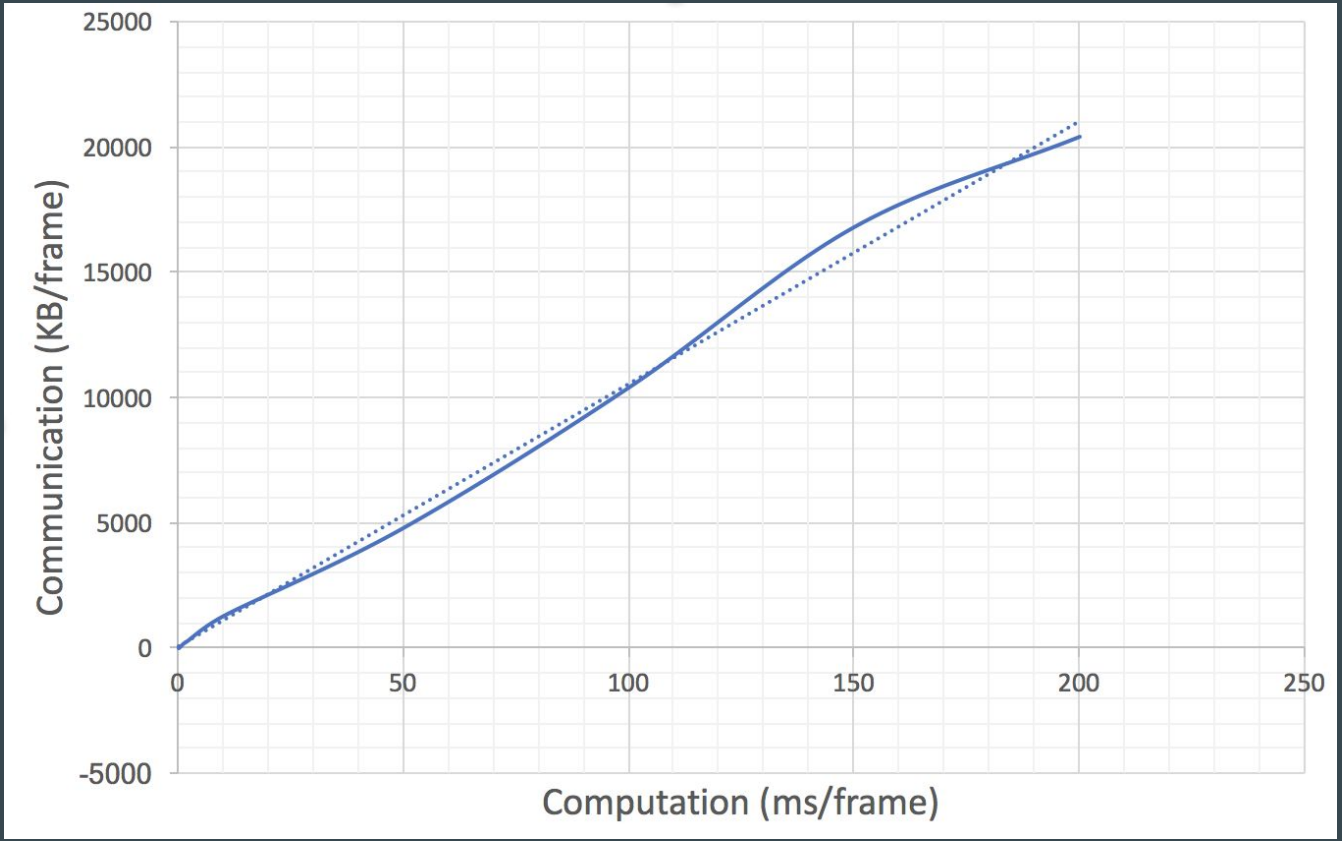
Question:

- No.
- The separation adds the communication time to the whole process. If the communication time is too significant compared to the actual original computation time of the interactions, there will be no reasonable speedup.

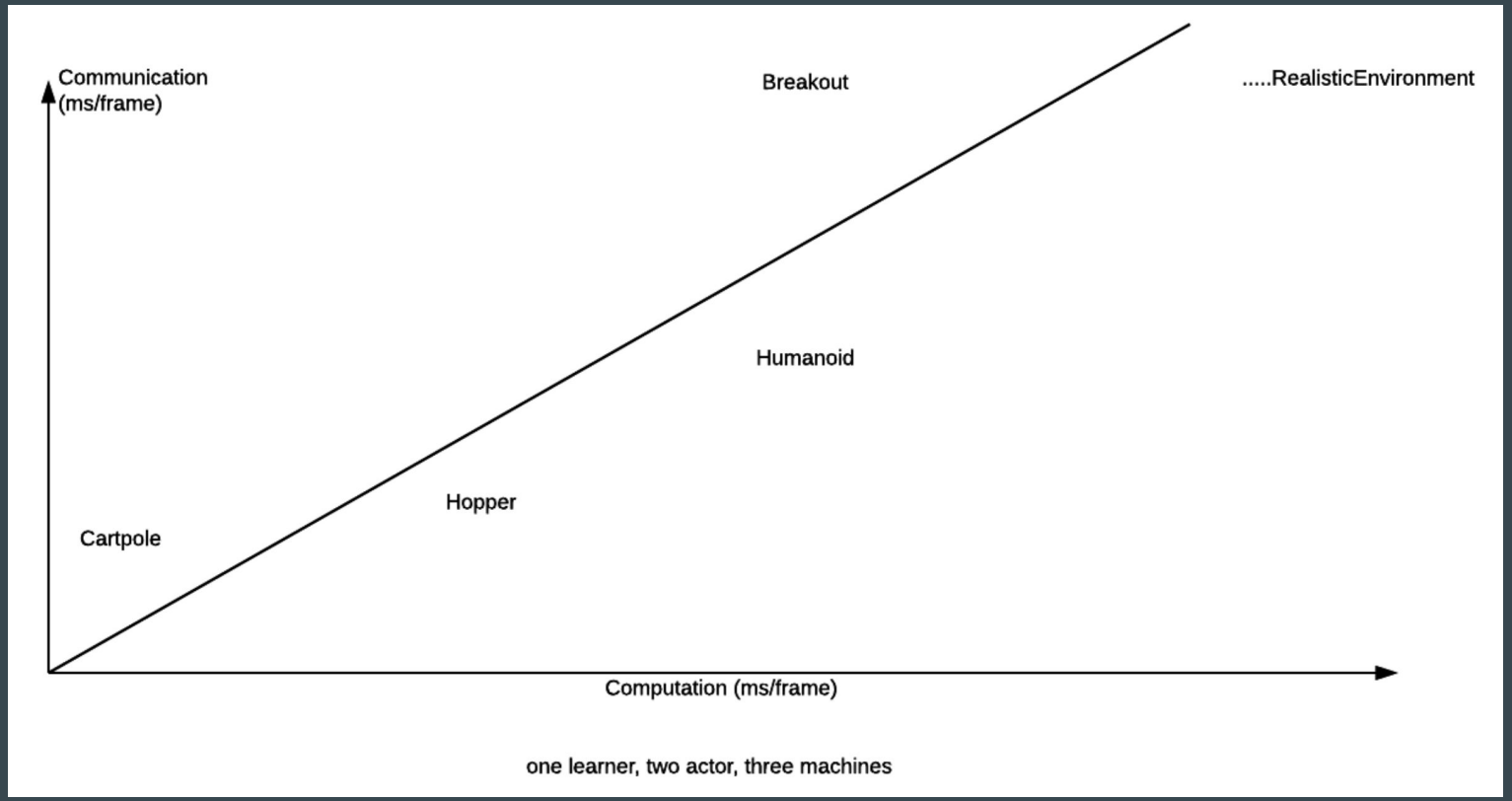
Simulation

- We don’t have enough real tasks to investigate this problem.
- Hence we simulated the process, use sleep() to simulate the computation time at actors to see a relation among communication time, computation time and speed up.

Simulation


25000
20000
15000
10000
5000
0
-5000
Communication (KB/frame)
0
50
100
150
200
250
Computation (ms/frame)

Simulation

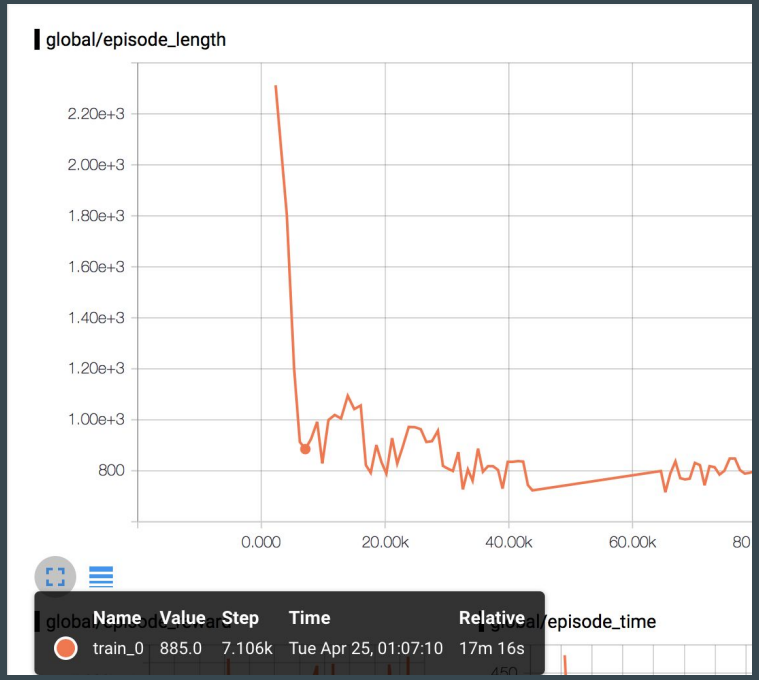

Communication
(ms/frame)
Breakout
.....RealisticEnvironment
Humanoid
Hopper
Cartpole
Computation (ms/frame)
one learner, two actor, three machines

Experiment Background

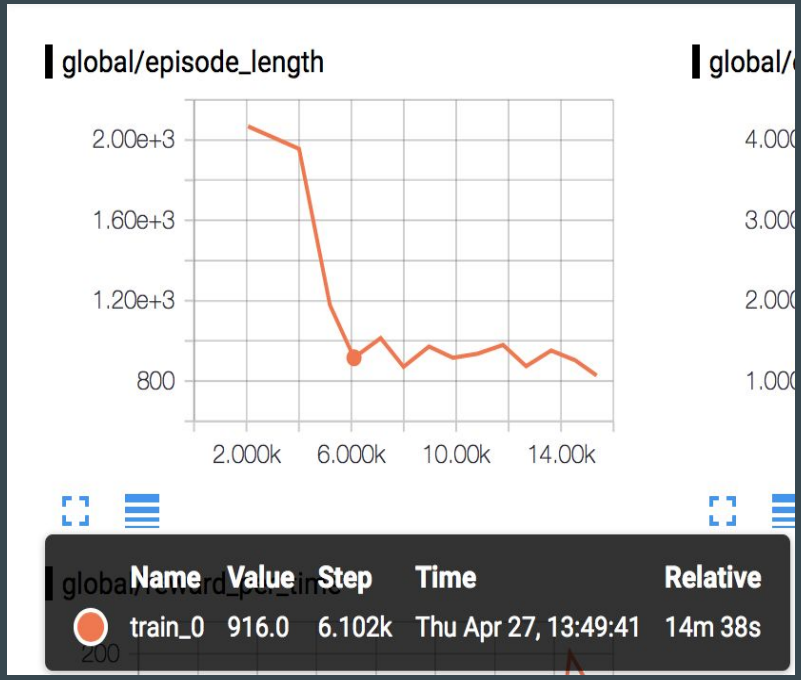
- A3C (Asynchronous Advantage Actor-Critic) Algorithm: An open source learning algorithm for Deep Reinforcement Learning tasks, released by Google DeepMind group.
- Neonrace game: A video game which gives us a car-driving task.

Multi-actor Result

- Original: actor not separated

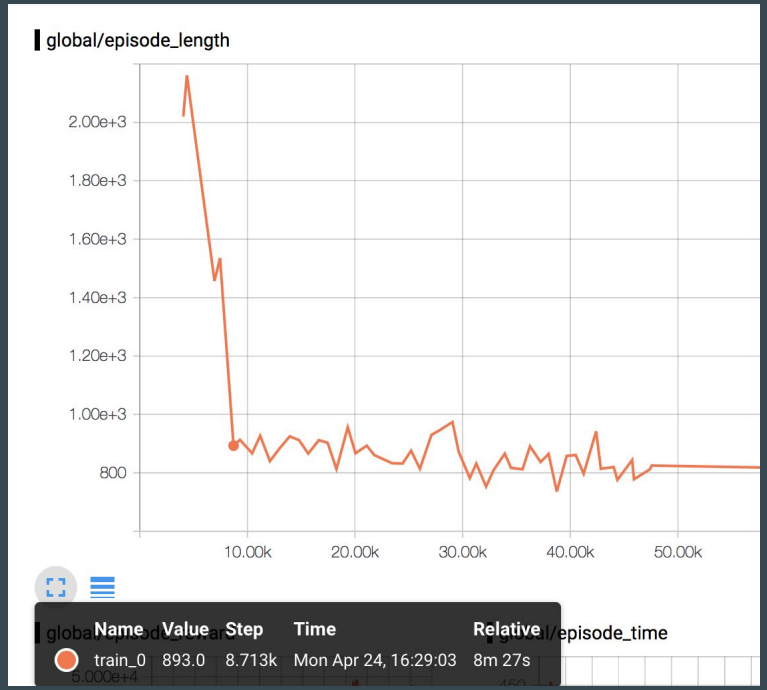


- One actor on separate machine

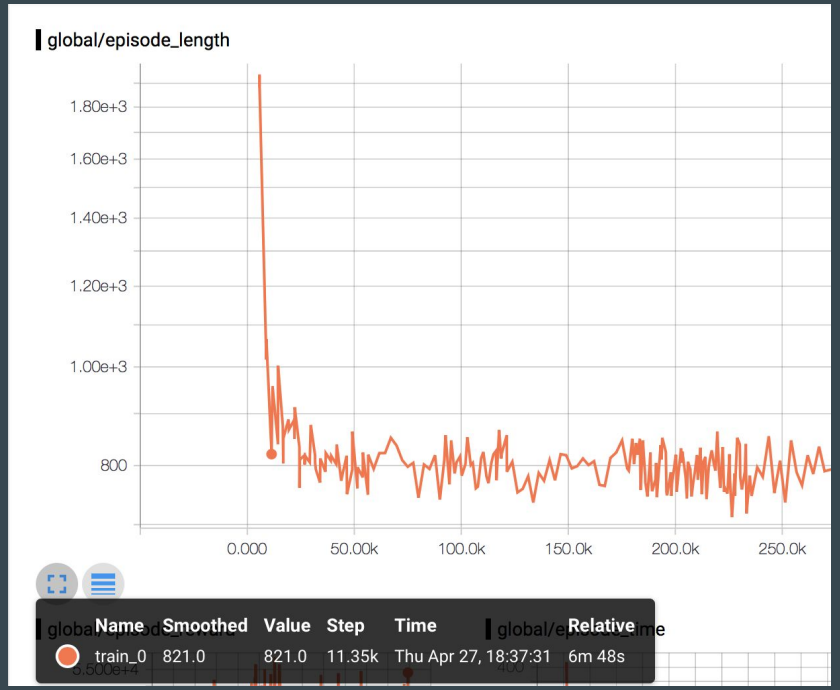


Multi-actor Result

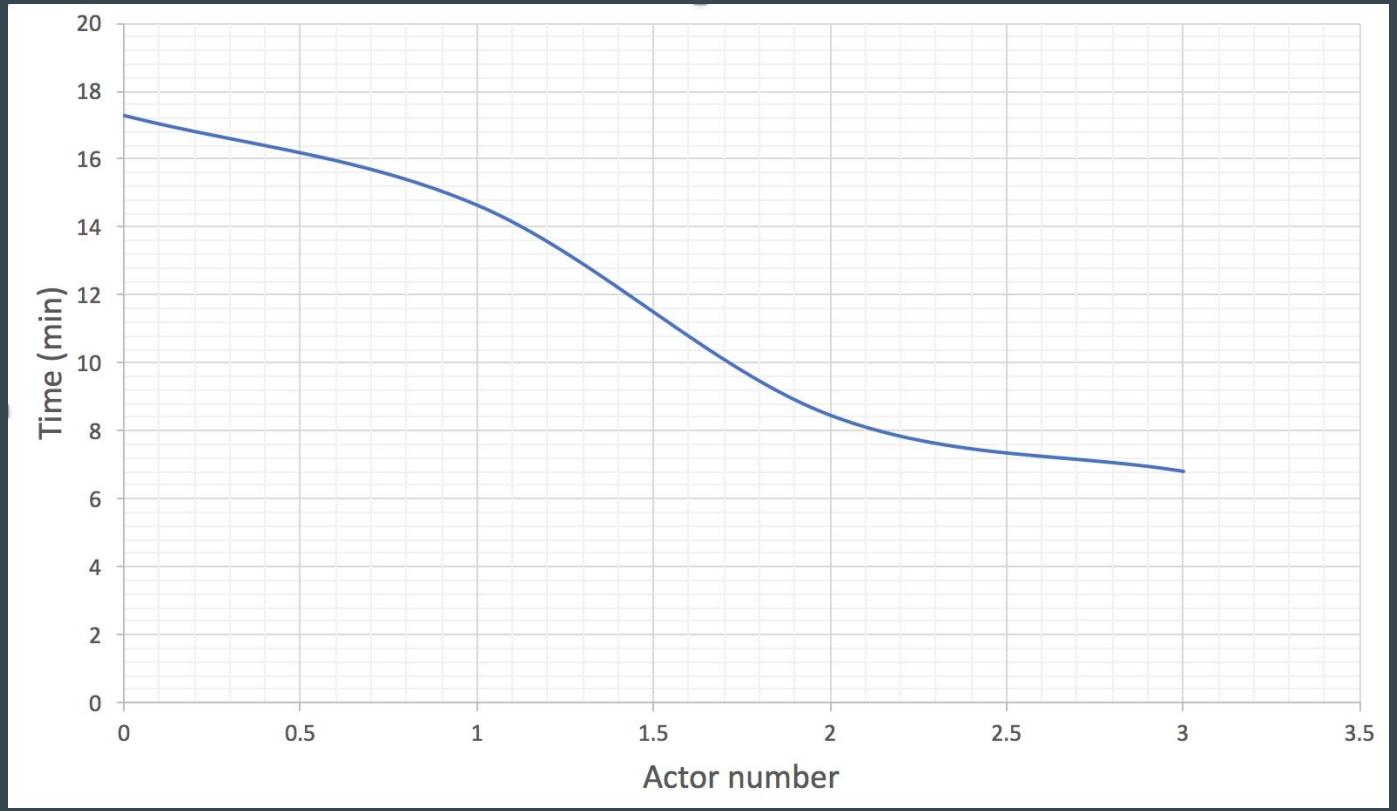
- Two actors



- Three actors

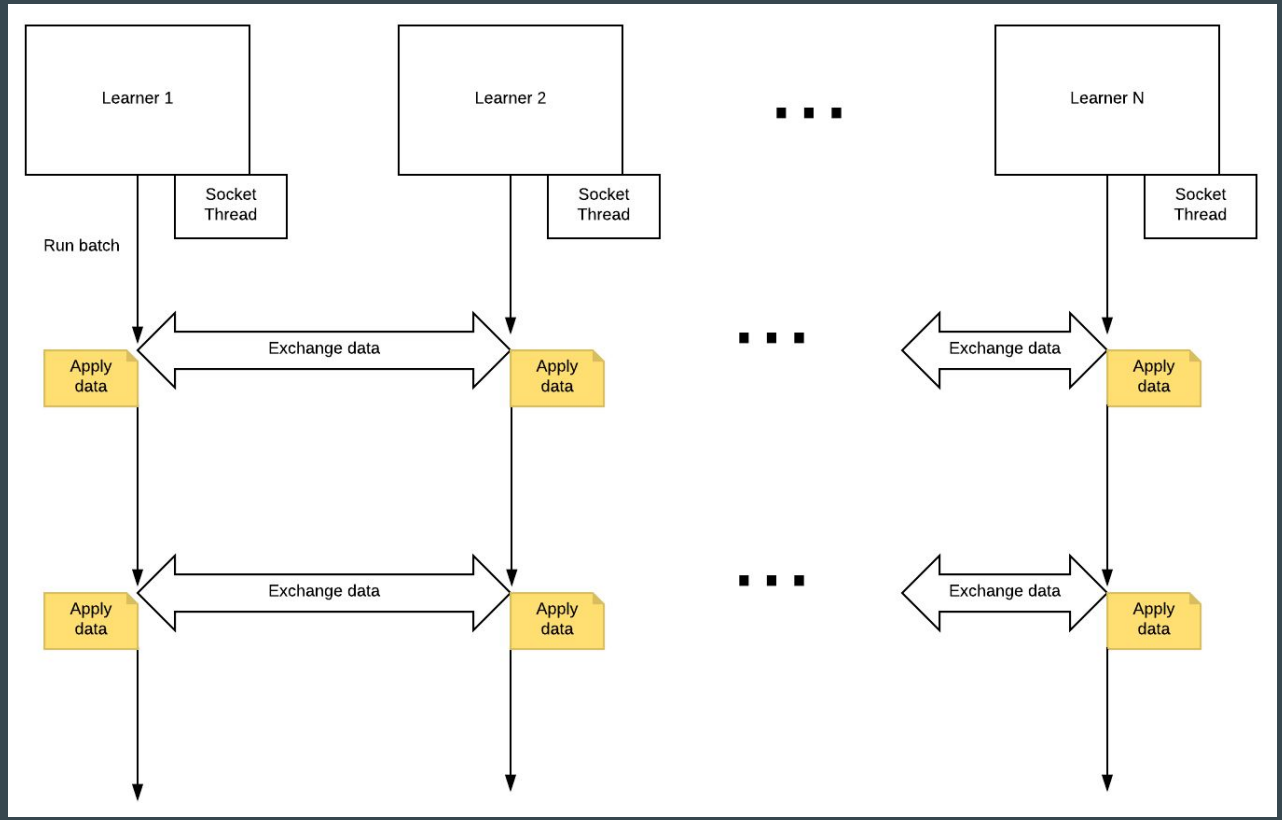


Multi-actor Result



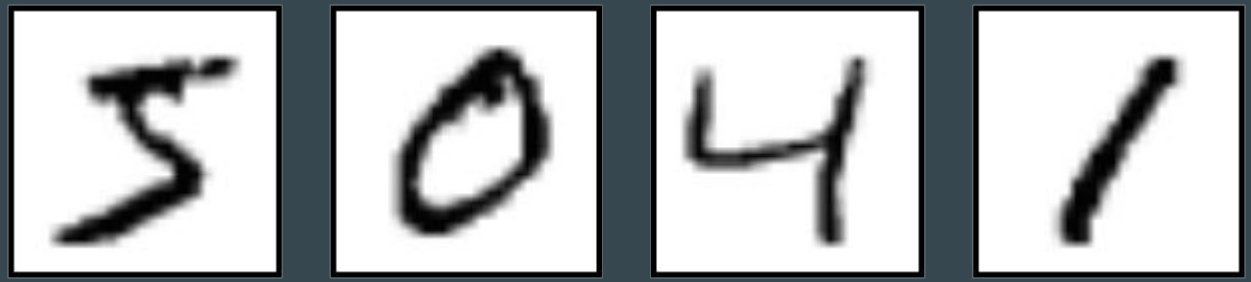
Multi-Learner Training

P2P Multi-learner Implementation



MNIST Task

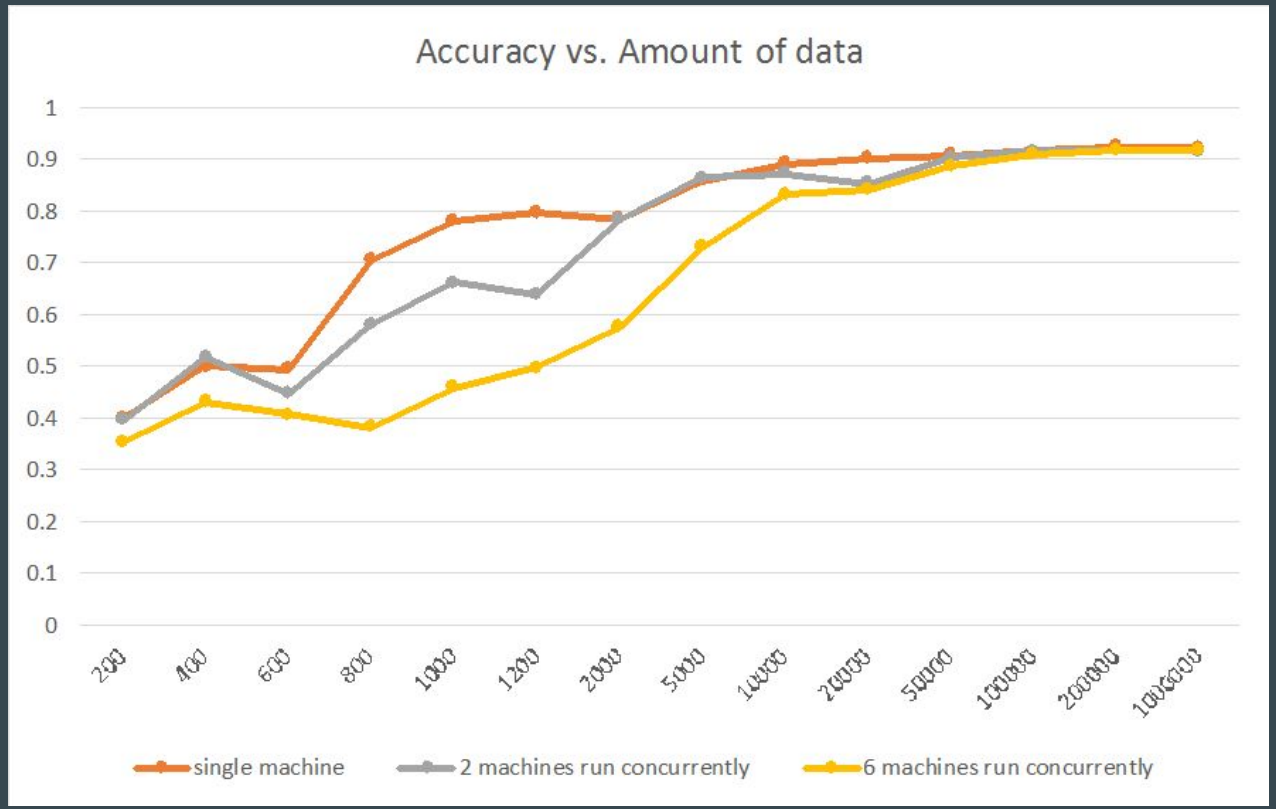
- large database of handwritten digits
- for training image processing and machine learning systems



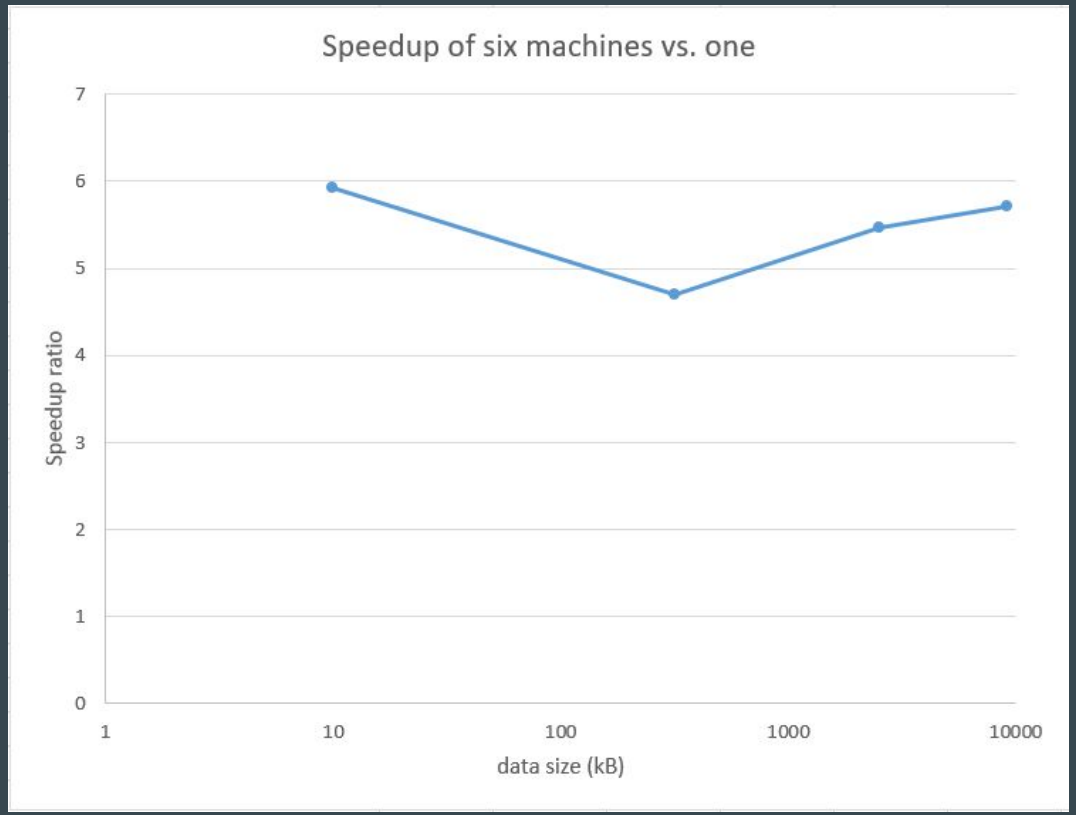
- labels for each image

Our task: train a model to look at images and predict what digits they are

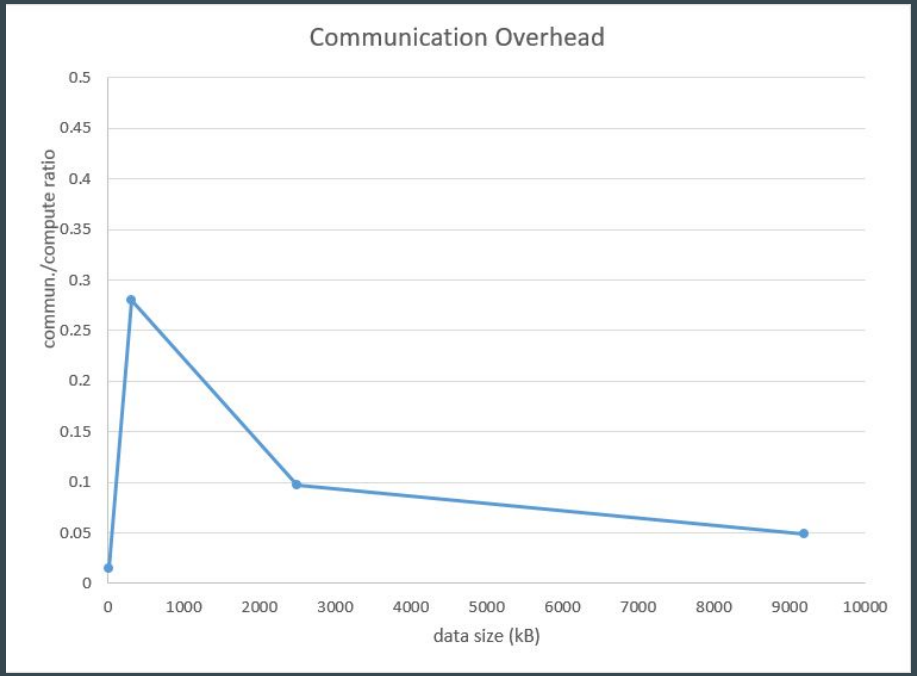
MNIST Results

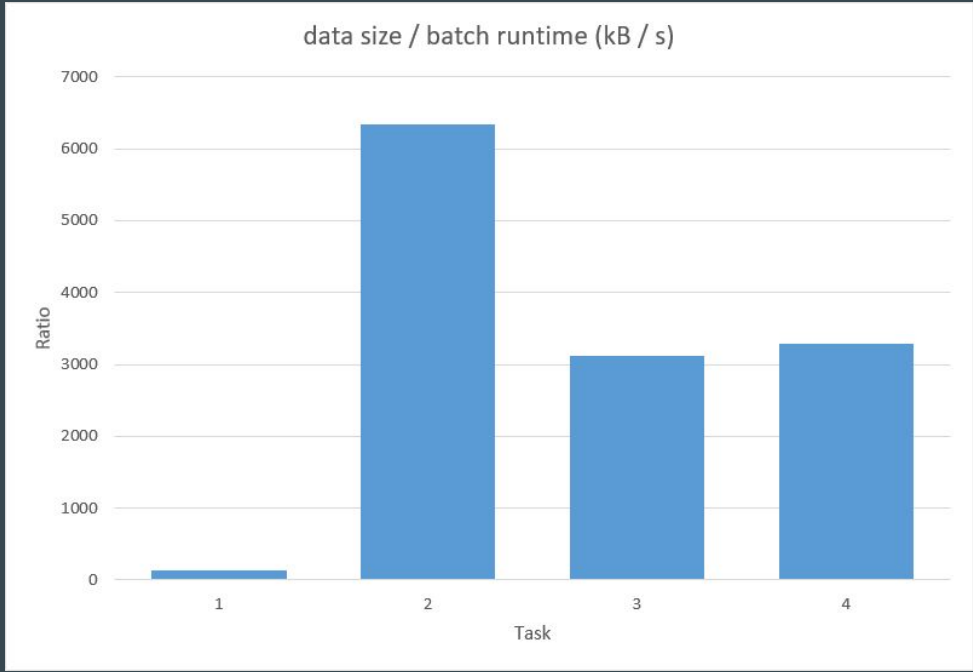


Computer Vision Tasks on Tensorflow



Why?


Communication Overhead
commun./compute ratio
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
data size (kB)


data size / batch runtime (kB / s)
Ratio
7000
6000
5000
4000
3000
2000
1000
0
1
2
3
4
Task

Integration into NeonRace

Issue 1: Message Loss

We have a message loss rate of 1.5-2%. Message loss is a serious issue here, since we are fragmenting each matrix data into ~1600 packets.

Fix: python multicast library (unreliable) $\Rightarrow$ Spread python wrapper (reliable)

15% slower, but it's okay

Issue 2: Learner Divergence

We can only synchronize as frequent as one data exchange per 10 steps. Otherwise, the communication overhead will be too much. Learners diverge as learning goes on, and even with weight exchanges they cannot converge.

English-Chinese Analogy

Fix: Use a Parameter Server (storage object) provided by Tensorflow to synchronize a central model periodically. This is a temporal fix, and ideally we should come up with better synchronization methods.

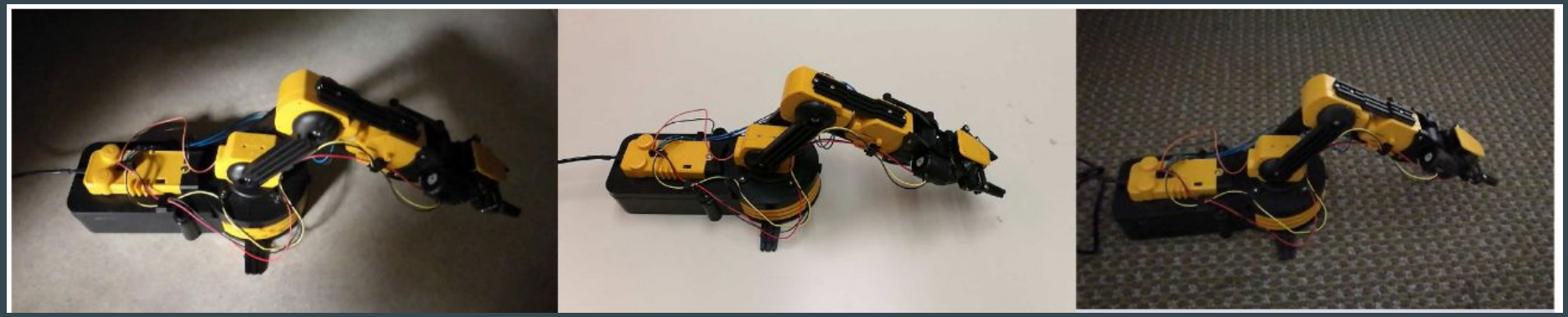
Cominatination of the multi-learner and multi-actor system

The adaptation between real-virtual domains

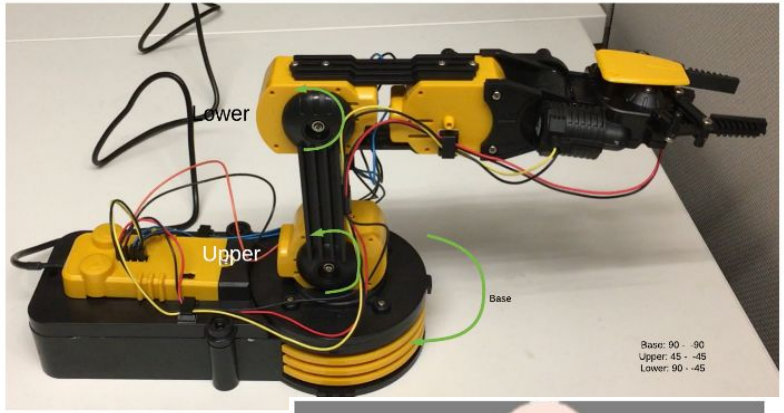
Domain adaptation

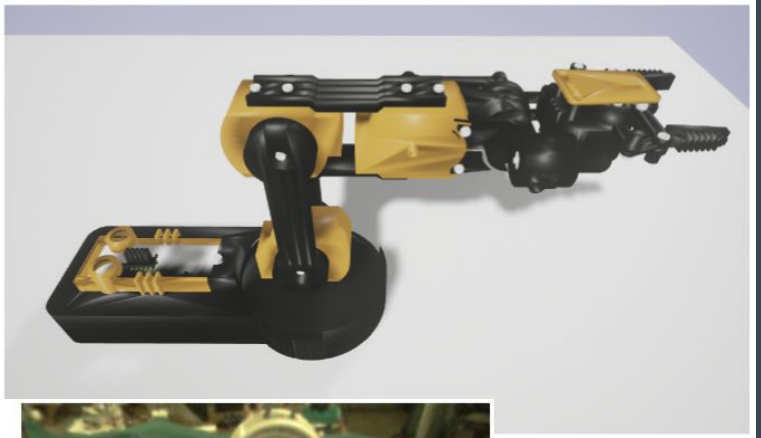
Whether the model trained in the virtual world be able to adapt to the real world?

Yes and No.

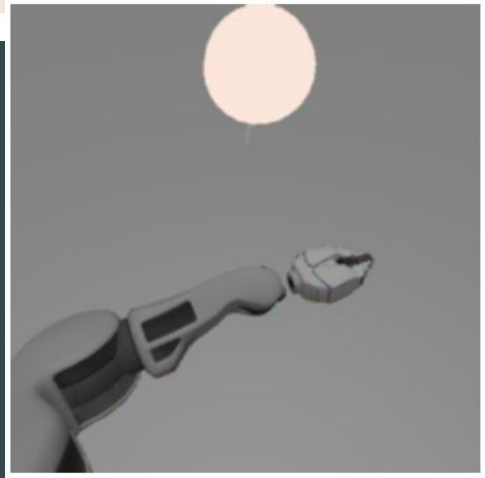


$30



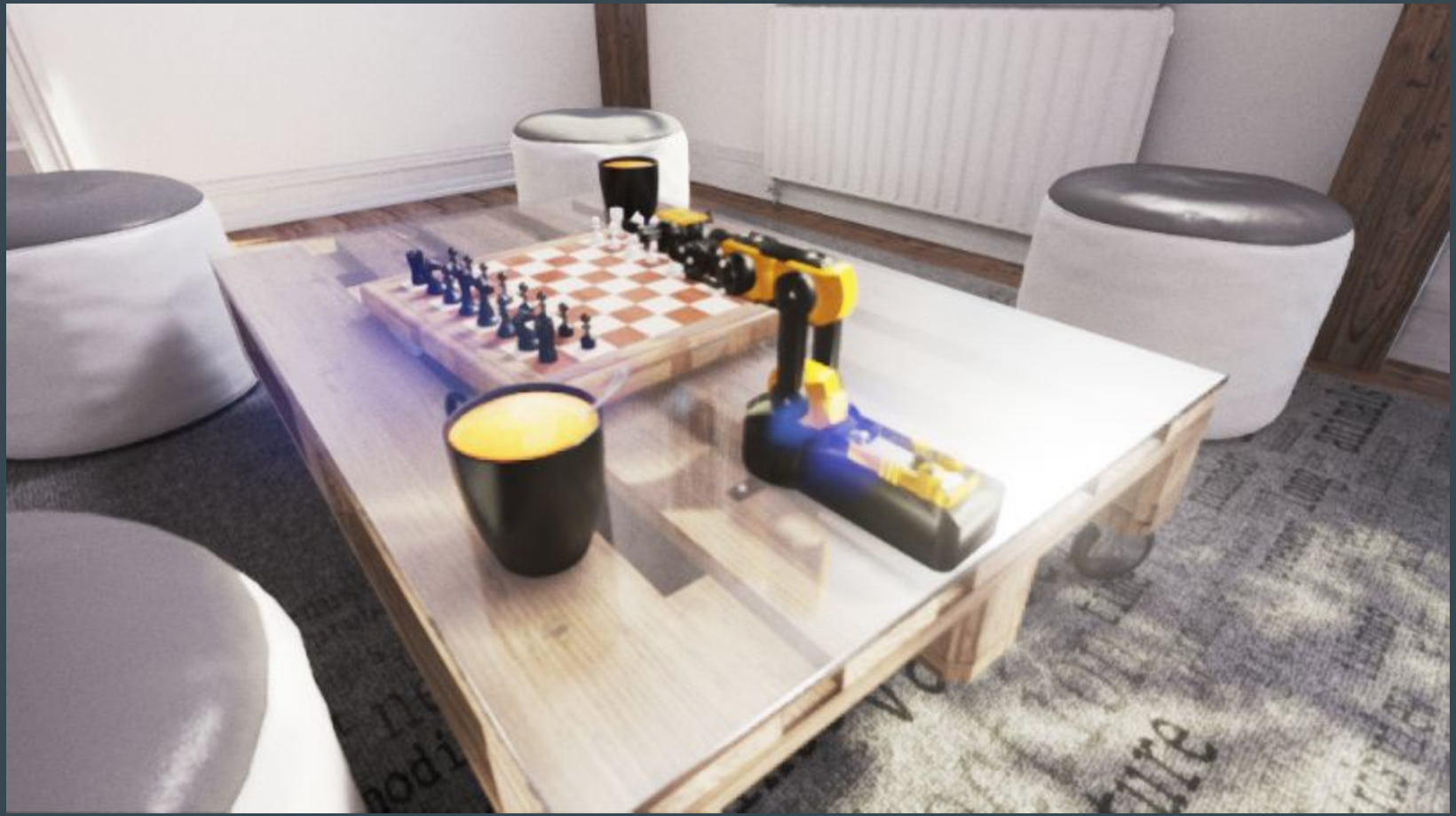


$300,000
[1] [2]



[1] http://www.willowgarage.com/pages/pr2/order
[2] Tzeng, Eric, et al. "Adapting deep visuomotor representations with weak pairwise constraints." Workshop on the Algorithmic Foundations of Robotics (WAFR). 2016.

A virtual + real robot arm platform
For domain-adaptation research
(a side product)




EPIC
GAMES

Two components in the platform

- A low-cost real robot arm
  - No sensor, five motors, inaccurate motion, inexpensive
- A realistic virtual arm
  - Can be placed into many realistic virtual environments and interact with objects
  - Similar appearance to the real arm
  - The appearance can be controlled

Download link: https://cs.jhu.edu/~qiuwch/RoboArm.zip
Purchase link: https://www.amazon.com/OWI-OWI-535-Robotic-Arm-Edge/dp/B0017OFRCY

A quick demo

Figure 6: Images from different virtual domains with different upper arm poses. From left to right are random color, random lighting and random camera position

	real	basic	bright	cam	rgb	rgb1	light
Basic	0.33	0.92	0.20	0.31	0.38	0.34	0.22
Random Color	0.09	0.19	0.30	0.31	0.66	0.60	0.37
Random Lighting	0.42	0.78	0.63	0.67	0.26	0.24	0.87

Table 2: Virtual domain adaptation measured by 10 degree accuracy

10 Degree accuracy: Prediction is considered correct, if the error is within 10 degrees.

Other progress with Realistic Virtual Environments

Deployment of Realistic Virtual Environments

- Nvidia-docker
  - Make realistic virtual environments can be easily deployed to Linux servers
- Headless linux server
  - Make it possible to use a Linux server farms to run many rendering tasks at the same time

Conclusion

Conclusion

- Distributed System + Computer Vision (AI) + Virtual Reality
- Multi-actor provides linear speedup
- Multi-learner provides linear speedup on non-complex tasks
- Learner divergence issue

Future work

- Domain adaptation research based on the robot arm
- Better synchronization methods for multi-learner